Documentation of ChatGPT Support for Advanced BA Project

Disclaimer: This report was also generated using Generative AI to collect prompts used in the project.

Group 25

April 25, 2025

Contents

1 Git/GitHub Issues & CI/CD Setup

Query

"Why does this not work in .gitignore when I specify `models/*` to ignore my model weights?"

Answer

Git does not automatically stop tracking files that were already committed, even if they are later added to `.gitignore`. To stop tracking a file that has already been committed, use:

```
git rm --cached models/your_model_file.pth
git commit -m "Remove large model file from repo"
```

This removes the file from version control without deleting it locally. The `.gitignore` then prevents future commits of this file.

Query

"I get the error 'No module named advanced_ba_project' when running my tests in GitHub Actions. What should I do?"

Answer

The project must be installed as a package in the GitHub Actions environment. Use:

```
pip install -e .
```

This allows Python to recognize the `advanced_ba_project` module using the `pyproject.toml` configuration.

Query

"How should my GitHub Actions workflow look for running unit tests with coverage?"

Answer

Here is the correct GitHub Actions configuration:

```yaml
name: Unit Tests

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.12", "3.11"]

    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: ${{ matrix.python-version }}
          cache: 'pip'

      - name: Install dependencies
        run: |
          python -m pip install -U pip setuptools wheel
          pip install -r requirements.txt
          pip install -e .

      - name: Run tests with coverage
        run: |
          coverage run --source=advanced_ba_project -m pytest
    tests/
          coverage report -m
```

Query

"I got an error with `export HYDRA_FULL_ERROR=1` on Windows runners in my GitHub workflow. How do I fix that?"

Answer

The `export` command is Unix-specific and does not work on Windows. To avoid this problem and support all operating systems, set environment variables like this:

```yaml
- name: Run tests & coverage
  env:
```

```
    HYDRA_FULL_ERROR: 1
  run: |
    coverage run --source=advanced_ba_project -m pytest tests/
    coverage report -m
```

This method works across Linux, macOS, and Windows runners.

Query

"How do I avoid counting the test files when measuring coverage?"

Answer

Add the following configuration to your `pyproject.toml`:

```
[tool.coverage.run]
omit = ["tests/*"]
```

This ensures that coverage measurement only includes your project code, not the test files themselves.

Query

"I can't push my model file to GitHub because it exceeds 100MB. What should I do?"

Answer

GitHub has a 100MB limit for individual files. To avoid this:

1. Remove the large file from Git tracking:

   ```
       git rm --cached models/your_model_file.pth
       git commit -m "Remove large model file"
   ```

2. Add the file pattern to your `.gitignore`:

   ```
       models/*.pth
   ```

3. Optionally, use Git LFS (Large File Storage): https://git-lfs.github.com/

Query

"How can I cache my Python dependencies in GitHub Actions?"

Answer

In your workflow, use the cache option provided by `actions/setup-python`:

```
- uses: actions/setup-python@v5
  with:
    python-version: ${{ matrix.python-version }}
    cache: 'pip'
```

This speeds up your CI runs by caching installed dependencies between builds.

2 Model Design, Architecture, and Evaluation

Query

"Can you help me understand the design of my U-Net model and why it looks like this?"

Answer

Your U-Net model follows an encoder-decoder architecture with skip connections between corresponding encoder and decoder layers. This design allows the model to:

- Downsample the input image to extract features (encoder).
- Upsample to reconstruct the segmentation mask (decoder).
- Use skip connections to combine low-level spatial information with high-level features, improving accuracy.

Structure Overview:

- Contracting blocks with convolution, batch normalization, ReLU activation, and dropout.
- Bottleneck layer at the deepest part of the network.
- Expanding blocks with upsampling and concatenation of encoder features.
- Final 1x1 convolution to reduce channels to the output class.

Query

"How does the forward pass in my U-Net work?"

Answer

The forward pass performs:

1. Encoding the input through several downsampling layers.
2. Processing through the bottleneck layer.
3. Decoding through upsampling, concatenating with encoder features (skip connections).
4. Final 1x1 convolution to output the prediction mask.

Query

"How should I describe the skip connections in U-Net?"

Answer

Skip connections directly link each encoder layer to its corresponding decoder layer. This helps:

- Preserve spatial information lost during downsampling.
- Improve localization performance.
- Allow gradients to flow more easily during backpropagation.

Query

"How do I explain the choices for dropout and batch normalization in my model?"

Answer

Batch Normalization helps stabilize training and allows higher learning rates by normalizing layer inputs. **Dropout** adds regularization by randomly dropping units during training, reducing the chance of overfitting.

In your U-Net:

- Batch normalization is applied after convolutions.
- Dropout is used in the contracting blocks and bottleneck, preventing overfitting.

Query

"What input and output sizes should my U-Net handle?"

Answer

The model is designed to handle inputs of shape:

```
(batch_size, 3, img_dim, img_dim)
```

and produces outputs of:

```
(batch_size, 1, img_dim, img_dim)
```

matching the size of the segmentation masks.

Query

"Can you explain why we use a 1x1 convolution in the final layer of the U-Net?"

Answer

The 1x1 convolution reduces the number of channels from the last decoder layer to the desired output channels (e.g., 1 for binary segmentation). It:

- Acts as a linear classifier at each pixel location.
- Maintains spatial resolution.
- Is computationally efficient.

Query

"What activation function should I use in the output layer?"

Answer

Since your task is binary segmentation:

- Use `sigmoid` activation on the output.
- Follow with a threshold (e.g., 0.5) to obtain binary masks.

Query

"What loss function should I use for segmentation tasks like this?"

Answer

The recommended approach is a combination of:

- Binary Cross-Entropy (BCE) loss: Stable pixel-wise classification.
- Dice loss: Focuses on overlap between predicted masks and ground truth.

The combination balances pixel-level accuracy and global overlap quality.

Query

"How do I evaluate the performance of my segmentation model?"

Answer

Common evaluation metrics for segmentation include:

- Accuracy
- Precision
- Recall
- F1 Score
- Intersection over Union (IoU)

You can also visualize:

- Training and validation loss curves.
- Example predictions vs. ground truth masks.

Query

"Can you give me an example of how to visualize the model predictions?"

Answer

Here's a minimal example for plotting predictions:

```python
import matplotlib.pyplot as plt
import torch

def visualize_prediction(model, image_tensor):
    model.eval()
    with torch.no_grad():
        pred = model(image_tensor.unsqueeze(0))
        pred = torch.sigmoid(pred).squeeze().cpu().numpy()
    plt.imshow(pred > 0.5, cmap='gray')
    plt.title('Predicted Mask')
    plt.axis('off')
    plt.show()
```

This helps qualitatively assess your model's performance.

3 Loss Functions and Optimizer Setup

Query

"What loss function should I use for my segmentation model?"

Answer

The project uses a **combined Dice + Binary Cross-Entropy (BCE) loss**, often referred to as `DiceBCELoss`. This combination was chosen because:

- BCE loss focuses on pixel-wise classification (each pixel as forest/non-forest).
- Dice loss emphasizes overlap between predicted and true masks, which helps especially when there is class imbalance (small forest regions vs. large background).

Query

"How is the Dice + BCE loss implemented?"

Answer

The total loss is calculated as:

$$\text{Loss} = \text{BCE Loss} + \text{Dice Loss}$$

The Dice loss formula is:

$$\text{Dice Loss} = 1 - \frac{2 \cdot (\text{intersection}) + \epsilon}{(\text{sum of predictions}) + (\text{sum of targets}) + \epsilon}$$

where ϵ is a small constant (e.g., $1e-6$) to avoid division by zero.

Query

"Why use both BCE and Dice loss together?"

Answer

BCE loss alone can struggle with imbalanced classes where the background dominates. Dice loss compensates for this by focusing on the overlap area between predictions and targets. Combining the two leverages:

- Stable convergence from BCE.
- Overlap sensitivity from Dice.

Query

"How do I set up the optimizer in PyTorch for my model?"

Answer

The optimizer used in the project is **Adam**. This optimizer is well-suited for segmentation tasks due to its adaptive learning rates. Additionally, `weight_decay` (L2 regularization) is applied to help avoid overfitting.

Example setup:

```
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=cfg.hyperparameters.learning_rate,
    weight_decay=cfg.hyperparameters.weight_decay
)
```

Query

"Should I use a learning rate scheduler for this model?"

Answer

Yes, the project uses `ReduceLROnPlateau`. This scheduler:

- Monitors the validation loss.
- Reduces the learning rate by a factor (e.g., 0.5) if no improvement is seen after a certain number of epochs (patience).

Example setup:

```
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode='min',
    factor=0.5,
    patience=3,
    verbose=True
)
```

Query

"Can you show me the full setup for the loss function, optimizer, and scheduler?"

Answer

Here's the complete setup used in your project:

```
criterion = DiceBCELoss()

optimizer = torch.optim.Adam(
    model.parameters(),
    lr=cfg.hyperparameters.learning_rate,
    weight_decay=cfg.hyperparameters.weight_decay
)

scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode='min',
    factor=0.5,
    patience=3,
    verbose=True
)
```

Query

"What hyperparameters should I tune for the optimizer and scheduler?"

Answer

Key hyperparameters tuned during your sweeps:

- `learning_rate`: Tested values like {0.001, 0.0001}.
- `weight_decay`: {0.001, 0.0001}.
- `scheduler factor`: Commonly 0.5.
- `scheduler patience`: Usually set to 3 epochs.

4 Model Testing, Unit Tests, and Coverage

Query

"How can I write a simple test for the UNet model?"

Answer

The basic test for the U-Net model should verify:

- Whether the model can be instantiated without errors.
- Whether the forward pass runs successfully.
- Whether the output shape matches the expected shape.

Example Unit Test Code

```
import torch
from advanced_ba_project.model import UNet

def test_unet_forward_pass():
    model = UNet(in_channels=3, out_channels=1)
    x = torch.randn((1, 3, 128, 128))
    out = model(x)
    assert out.shape == (1, 1, 128, 128), "Output shape is incorrect"
```

Query

"How can I improve this test to check the model structure as well?"

Answer

For a more robust check, you can test both the structure and the forward pass:

```python
def test_unet_structure_and_forward():
    model = UNet(in_channels=3, out_channels=1)
    assert model.in_channels == 3
    assert model.out_channels == 1
    assert hasattr(model, 'encoder'), "Encoder not found in model"
    assert hasattr(model, 'decoder'), "Decoder not found in model"

    x = torch.randn((1, 3, 256, 256))
    out = model(x)
    assert out.shape == (1, 1, 256, 256), "Incorrect output shape"
```

Query

"Why do I get an import error: 'No module named advanced_ba_project' during testing?"

Answer

This happens if your project is not properly installed as a Python package. The solution is to include:

```
pip install -e .
```

This makes `advanced_ba_project` discoverable by Python and your test runner.

Query

"How do I run my tests with coverage tracking?"

Answer

You can use `pytest` together with `coverage.py`:

```
coverage run --source=advanced_ba_project -m pytest tests/
coverage report -m
```

Query

"How should my GitHub Actions workflow be configured to run these tests?"

Answer

Your CI workflow should:

- Install all dependencies via `requirements.txt`.
- Install your package using `pip install -e .`.
- Run `coverage` with `pytest`.

Example GitHub Actions Workflow

```
name: Unit Tests

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.12", "3.11"]

    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: ${{ matrix.python-version }}
          cache: 'pip'

      - name: Install dependencies
        run: |
          python -m pip install -U pip setuptools wheel
          pip install -r requirements.txt
          pip install -e .
```

```
      - name: Run tests with coverage
        run: |
          coverage run --source=advanced_ba_project -m pytest
    tests/
          coverage report -m
```

Query

"Why did I get errors when using `export` in my GitHub Actions on Windows?"

Answer

The `export` command works on Linux and macOS but not on Windows. For GitHub Actions to work across all operating systems, use:

```
env:
  HYDRA_FULL_ERROR: 1
```

This is cross-platform and safe on Ubuntu, macOS, and Windows runners.

Query

"How do I prevent coverage from including my test files?"

Answer

In your `pyproject.toml`, add:

```
[tool.coverage.run]
omit = ["tests/*"]
```

Query

"Is it necessary to test performance metrics in my unit tests?"

Answer

No, unit tests should focus on:

- Correct model initialization.
- Forward pass correctness.

- Output shape validation.

Performance metrics (like accuracy or F1 score) should be evaluated separately in your training and evaluation scripts, not in your unit tests.

5 Dataset Loading, Indexing, and Augmentation

Query

"Why don't I get the same images when I use the same indices between `visualize_raw_and_predicted` and `visualize_raw_and_baseline`?"

Answer

The issue was that the file ordering in your dataset loader was not guaranteed to be consistent across different runs. When using `Path.glob("*.jpg")` or similar functions, the file ordering depends on the filesystem and is not sorted by default.

Solution

Always explicitly sort your list of image files to make indexing reliable:

```
image_paths = sorted(Path(image_dir).glob("*.jpg"))
```

This ensures that index `0` always corresponds to the same image across both functions.

Example Fix for RoboflowTreeDataset

Replace:

```
self.samples = self._build_samples()
```

with:

```
self.samples = sorted(self._build_samples())
```

Query

"Is there any way I can sort the dataset externally without changing the RoboflowTreeDataset class?"

Answer

If you do not want to change the dataset class itself, you can still enforce sorting externally by sorting the `indices` list when calling your visualization functions:

```
indices = sorted(indices)
```

However, it is safer and more reliable to sort the file paths at the source (within the dataset loader).

Query

"What kind of augmentations were used during training?"

Answer

The project applied basic data augmentations through the `Augmentation` class. These included:

- Random horizontal flips (probability configurable).
- Random vertical flips (probability configurable).
- Random rotations by 90°, 180°, or 270° (probability configurable).

Augmentation Class Example

```
class Augmentation(Dataset):
    def __init__(self, dataset, flip_prob=0.5, rotate_prob
   =0.3):
        self.dataset = dataset
        self.flip_prob = flip_prob
        self.rotate_prob = rotate_prob

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        image, mask = self.dataset[idx]

        if random.random() < self.flip_prob:
            image = torch.flip(image, [2])
            mask = torch.flip(mask, [2])

        if random.random() < self.flip_prob:
            image = torch.flip(image, [1])
```

```python
            mask = torch.flip(mask, [1])

        if random.random() < self.rotate_prob:
            k = random.choice([1, 2, 3])
            image = torch.rot90(image, k, [1, 2])
            mask = torch.rot90(mask, k, [1, 2])

        return image, mask
```

Query

"How were the training, validation, and test sets split?"

Answer

The dataset was split into:

- 80% training.
- 10% validation.
- 10% test.

This was done using `torch.utils.data.random_split` with a manual seed for reproducibility.

Query

"How do I ensure that the validation and test sets are consistent between runs?"

Answer

Set the random seed explicitly before splitting:

```python
torch.manual_seed(seed)
train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size
train_set, val_set, test_set = random_split(dataset, [
    train_size, val_size, test_size])
```

This ensures the splits remain consistent as long as the dataset order remains consistent.

Key Takeaways

- Always sort file paths when loading data.
- Apply a fixed seed when splitting datasets to keep validation and test sets stable.
- Use augmentation to improve model generalization.

6 Image Normalization and Prediction on External API Images

Query

"Why does my model work well on the training images but not on the API-provided `.tif` images?"

Answer

The model was trained on images normalized with:

```
transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5,
    0.5])
```

However, the API images had pixel values outside of the expected range and were not scaled between 0 and 1 before normalization. This caused the model to receive data in a very different format from what it saw during training.

Solution

Ensure that all input images are first scaled to the `[0, 1]` range before applying normalization. The correct approach is:

```
img_array = img_array.astype(np.float32)
img_array = (img_array - img_array.min()) / (img_array.max()
    - img_array.min() + 1e-8)
```

Query

"Can I simply scale the API images between 0 and 1 and then apply the same normalization?"

Answer

Yes, this is the recommended approach. First scale the images to `[0, 1]`, and then apply the exact same normalization used in your training process to ensure consistency.

Corrected Image Loader Function

```
def load_tif_image(path, img_dim=256):
    img = Image.open(path)
    img_array = np.array(img)

    # Handle multi-channel images or grayscale
    if img_array.ndim == 3 and img_array.shape[2] > 3:
        img_array = img_array[:, :, :3]
    if img_array.ndim == 2:
        img_array = np.stack([img_array] * 3, axis=-1)

    # Scale to [0, 1]
    img_array = img_array.astype(np.float32)
    img_array = (img_array - img_array.min()) / (img_array.max() - img_array.min() + 1e-8)

    img_pil = Image.fromarray((img_array * 255).astype(np.uint8))
    transform = transforms.Compose([
        transforms.Resize((img_dim, img_dim)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    ])
    return transform(img_pil)
```

Query

"Why did scaling fix the issue?"

Answer

The neural network expects input values to be normalized around zero with a standard deviation of one. Without scaling, the `.tif` images had unpredictable ranges (sometimes 0–65535, not 0–255), so the normalization step failed to align them properly with the trained data distribution.

Scaling ensures the model sees images in the same format as during training.

Query

"Could histogram matching or equalization work instead of scaling?"

Answer

Histogram matching or equalization could help if the issue was related to contrast or brightness differences between datasets. However, the primary problem here was not contrast—it was the range of pixel values. Simple scaling to `[0, 1]` is sufficient and more consistent with the training pipeline.

Debugging Insight

Plotting histograms of the pixel distributions from your API images and comparing them to your training set helped identify the scaling mismatch. Without proper scaling, the pixel distributions looked drastically different.

Observed Result

After applying scaling:

- Predictions on API images became significantly more stable.
- Although performance was still not perfect (due to dataset domain differences), the output masks were reasonable.

Additional Recommendation

For future work, consider:

- Including API-style images in your training dataset.
- Using domain adaptation techniques or augmentations to simulate such images during training.

7 Training, Hyperparameter Tuning, and Evaluation Metrics

Query

"How should I describe the hyperparameter tuning and training process?"

Answer

Hyperparameter tuning was performed using **Weights & Biases (wandb)** Sweeps with a grid search strategy. The main metric optimized was the **validation F1 score**.

Hyperparameters Tuned

The sweep explored the following parameter combinations:

- `batch_size`: {16, 32}
- `learning_rate`: {0.001, 0.0001}
- `weight_decay`: {0.001, 0.0001}
- `dropout_rate`: {0.1, 0.2, 0.3}
- `apply_augmentation`: {true, false}

Query

"How should I explain the data augmentation strategy used?"

Answer

Data augmentation was applied during training to improve generalization. The following augmentations were used:

- Random horizontal flips.
- Random vertical flips.
- Random rotations (90°, 180°, or 270°).

These augmentations were applied to both input images and their corresponding masks. The logic is implemented in the `Augmentation` class located in `data.py`.

Query

"How were the datasets split for training, validation, and testing?"

Answer

The project combined data from the **DeepGlobe** and **Roboflow** datasets with the following splits:

- 80% for training.
- 10% for validation.
- 10% for testing.

Query

"What learning rate scheduler was used?"

Answer

The **ReduceLROnPlateau** scheduler was used, monitoring the validation loss. If no improvement was observed for 3 epochs, the learning rate was reduced by a factor of 0.5.

Scheduler Example Code

```
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='min', factor=0.5, patience=3, verbose=True
)
```

Query

"How were the final results evaluated and reported?"

Answer

Performance evaluation included:

- Validation and test loss.
- Accuracy.
- Precision.
- Recall.

- F1 score.
- IoU (Intersection over Union).

Metrics Logging Example with wandb

```
wandb.log({
    "Validation Loss": val_loss,
    "Val Accuracy": metrics["accuracy"],
    "Val Precision": metrics["precision"],
    "Val Recall": metrics["recall"],
    "Val F1": metrics["f1"],
    "Val IoU": metrics["iou"],
}, step=epoch + 1)
```

Query

"How were the best hyperparameters determined?"

Answer

The sweep results were analyzed via Weights & Biases to identify the hyperparameter combination that maximized the validation F1 score. This combination was then selected for training the final model.

Observed Results

- Dropout rate and learning rate had significant impact on generalization.
- Applying augmentation generally improved the F1 score.
- Larger batch sizes worked better when GPU memory allowed.

8 General Debugging and Additional Helpful Tips

Query

"How can I get clearer error messages when using Hydra in my GitHub Actions setup?"

Answer

Hydra sometimes suppresses error output by default. To ensure that full stack traces are displayed, set:

```
HYDRA_FULL_ERROR=1
```

In GitHub Actions, for cross-platform compatibility (Linux, Windows, MacOS), use:

```
env:
  HYDRA_FULL_ERROR: 1
```

Query

"Why did my CI/CD pipeline break with the error: 'The term 'export' is not recognized as the name of a cmdlet...'?"

Answer

This occurred because the `export` command is specific to Unix-like shells. On Windows, use:

```
set HYDRA_FULL_ERROR=1
```

Or, for cross-platform workflows (recommended):

```
env:
  HYDRA_FULL_ERROR: 1
```

Query

"How can I avoid large model files causing GitHub push errors?"

Answer

GitHub has a 100MB file size limit. To avoid this issue:

- Untrack large files already committed:

  ```
      git rm --cached path/to/large_file.pth
      git commit -m "Remove large file from repo"

  ```

- Add these files to `.gitignore` to prevent future commits.
- Consider using **Git LFS (Large File Storage)**:

```
git lfs install
git lfs track "*.pth"
git add .gitattributes
git add path/to/large_file.pth
git commit -m "Track large model file with Git LFS"
```

Query

"What helped you quickly identify the issue with image normalization mismatches?"

Answer

The following debugging steps were effective:

- Plot histograms of pixel values for both training images and API images.
- Compare mean and standard deviation statistics between datasets.
- Visualize example images before and after normalization to confirm the preprocessing pipeline is consistent.

Query

"Any general tips for reproducibility and debugging?"

Answer

Good practices for reproducibility and debugging included:

- Setting random seeds for Python, NumPy, and PyTorch:

```
import random
import numpy as np
import torch

random.seed(42)
np.random.seed(42)
torch.manual_seed(42)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(42)
```

- Logging configuration parameters (e.g., via Hydra or wandb).
- Keeping training and evaluation environments consistent.

FROM DOWNWARDS THE PROMPTS ORIGINATE FROM ANOTHER CHAT HISTORY.

9 Visualization Techniques

Query

How can I visualize segmentation masks overlayed on original images?

Answer

Implemented a custom function with color-coded overlays:

```
def visualize_batch_from_dataset(dataset, num_images=5,
    subset_index=None):
    if subset_index is not None and isinstance(dataset, torch
    .utils.data.ConcatDataset):
        dataset = dataset.datasets[subset_index]
    indices = torch.randperm(len(dataset))[:num_images]
    images, masks = zip(*[dataset[i] for i in indices])
    for i in range(num_images):
        overlay[mask_np==1.0] = [0, 1, 0]  # Green
        overlay[mask_np==0.0] = [0.5, 0, 0]  # Dark red
```

Query

How can I choose which dataset to visualize within a `ConcatDataset`*?*

Answer

Added a `subset_index` parameter:

```
visualize_batch_from_dataset(val_loader.dataset, num_images
    =5, subset_index=0) # Forest
visualize_batch_from_dataset(val_loader.dataset, num_images
    =5, subset_index=1) # Roboflow
```

Query

What RGB value represents dark red for mask overlays?

Answer

Dark red is represented by:

```
overlay[non_forest_mask] = (139, 0, 0)
```

This color choice supports clarity and accessibility.

Query

How can I visualize segmented forest areas from two different datasets ("forest" and "roboflow") with consistent overlay coloring?

Answer

The following Python function provides a robust method to visualize segmentation masks overlaid on satellite images from either the Forest or Roboflow dataset. It includes options for setting random seeds for reproducibility or specifying exact indices for visualization.

```
from advanced_ba_project.data import ForestDataset,
    RoboflowTreeDataset
from pathlib import Path
import torch
import random
import numpy as np
import matplotlib.pyplot as plt
from torchvision import transforms
from matplotlib.patches import Patch

def plot_forest_overlay_grid_from_dataset(
dataset_name: str,
data_path: Path,
metadata_file: str = None,
image_dir: Path = None,
label_dir: Path = None,
num_images: int = 5,
img_dim: int = 256,
seed: int = 42,
indices: list = None
):
transform = transforms.Compose([
transforms.Resize((img_dim, img_dim)),
transforms.ToTensor(),
transforms.Normalize(mean=[0.5]*3, std=[0.5]*3)
])
target_transform = transforms.Compose([
transforms.Resize((img_dim, img_dim)),
```

```python
transforms.ToTensor()
])

if dataset_name == "forest":
    if metadata_file is None:
        raise ValueError("metadata_file must be provided for
    forest dataset.")
    dataset = ForestDataset(
        data_path=data_path,
        metadata_file=metadata_file,
        transform=transform,
        target_transform=target_transform
    )
elif dataset_name == "roboflow":
    if image_dir is None or label_dir is None:
        raise ValueError("image_dir and label_dir must be
    provided for roboflow dataset.")
    dataset = RoboflowTreeDataset(
        image_dir=image_dir,
        label_dir=label_dir,
        patch_size=img_dim,
        transform=transform,
        target_transform=target_transform
    )
else:
    raise ValueError("dataset_name must be either 'forest' or
     'roboflow'.")

if indices is None:
    torch.manual_seed(seed)
    random.seed(seed)
    indices = random.sample(range(len(dataset)), min(
    num_images, len(dataset)))
else:
    num_images = len(indices)

samples = [dataset[i] for i in indices]
fig, axs = plt.subplots(2, num_images, figsize=(4 *
    num_images, 8))

for i, (image, mask) in enumerate(samples):
    image = image.permute(1, 2, 0).numpy()
    image = (image * 0.5) + 0.5
    image = np.clip(image, 0, 1)

    mask = mask.squeeze().numpy()
    if mask.ndim == 3:
        mask = mask[0]

```

```python
    axs[0, i].imshow(image)
    axs[0, i].axis("off")
    axs[0, i].set_title(f"{dataset_name.capitalize()} Original {indices[i]}")

    overlay = image.copy()
    forest_color = np.array([0, 1, 0])
    nonforest_color = np.array([1, 0.3, 0])
    alpha = 0.5

    mask_bool = mask > 0.5
    overlay[mask_bool] = (1 - alpha) * overlay[mask_bool] + alpha * forest_color
    overlay[~mask_bool] = (1 - alpha) * overlay[~mask_bool] + alpha * nonforest_color

    axs[1, i].imshow(overlay)
    axs[1, i].axis("off")
    axs[1, i].set_title(f"{dataset_name.capitalize()} Overlay {indices[i]}")

legend_elements = [
    Patch(facecolor='green', edgecolor='black', label='Forest', alpha=0.5),
    Patch(facecolor='orangered', edgecolor='black', label='Non-Forest', alpha=0.5),
]
fig.legend(handles=legend_elements, loc='lower center', ncol=2, fontsize='large')
plt.tight_layout(rect=[0, 0.05, 1, 1])
plt.show()
```

Listing 1: Function to plot overlay grids from dataset

Usage Example

```python
plot_forest_overlay_grid_from_dataset(
dataset_name="roboflow",
data_path=None,
image_dir=Path("data/raw/roboflow/test/images"),
label_dir=Path("data/raw/roboflow/test/labelTxt"),
seed=100,
num_images=5
)

plot_forest_overlay_grid_from_dataset(
dataset_name="forest",
data_path=Path("data/raw/Forest Segmented"),
```

```
metadata_file="meta_data.csv",
indices=[0, 3, 7, 12, 25]
)
```

Query

How can I ensure that my quick sample visualizations use the same forest/non-forest overlay colors?

Answer

Use this streamlined function to visualize samples consistently across datasets, applying the same green and orange-red overlay scheme as in the main function.

```
from matplotlib.patches import Patch
import numpy as np
import matplotlib.pyplot as plt

def visualize_samples(dataset, num_samples=5, alpha=0.5):
fig, axes = plt.subplots(1, num_samples, figsize=(4 * num_samples, 4))
for i in range(num_samples):
image, mask = dataset[i]
image = image.permute(1, 2, 0).numpy()
image = (image * 0.5) + 0.5
image = np.clip(image, 0, 1)

    mask = mask.squeeze().numpy()
    if mask.ndim == 3:
        mask = mask[0]

    overlay = image.copy()
    forest_color = np.array([0, 1, 0])
    nonforest_color = np.array([1, 0.3, 0])
    mask_bool = mask > 0.5

    overlay[mask_bool] = (1 - alpha) * overlay[mask_bool] + alpha * forest_color
    overlay[~mask_bool] = (1 - alpha) * overlay[~mask_bool] + alpha * nonforest_color

    axes[i].imshow(overlay)
    axes[i].axis("off")
    axes[i].set_title(f"Sample {i+1}")

legend_elements = [
```

```
    Patch(facecolor='green', edgecolor='black', label='Forest
    ', alpha=alpha),
    Patch(facecolor='orangered', edgecolor='black', label='
    Non-Forest', alpha=alpha),
]
fig.legend(handles=legend_elements, loc='lower center', ncol
    =2, fontsize='large')
plt.tight_layout(rect=[0, 0.05, 1, 1])
plt.show()
```

Listing 2: Consistent overlay visualization for quick samples

Usage Example

```
visualize_samples(roboflow_train, num_samples=5)
visualize_samples(forest_dataset, num_samples=3)
```

10 Dataset Loading

Query

How should I organize my dataset for forest segmentation tasks in PyTorch? (image upload of current setup and I also already uploaded example code from already implemented dataloader)

Answer

Two datasets were used:

- **Forest Segmented**: Contains RGB satellite images and float-based binary masks (0.0 for non-forest, 1.0 for forest).
- **Roboflow**: Contains images with polygon annotations (converted into masks).

Folder structure:

```
data/
          raw/
                Forest Segmented/
                      images/
                      masks/
                      meta_data.csv
                roboflow/
                    train/
                    valid/
```

Query

How should the dataset classes be structured for loading both datasets?

Answer

Two custom dataset classes were implemented:

ForestDataset:

```python
class ForestDataset(Dataset):
    def __init__(self, data_path, metadata_file, transform=None, target_transform=None):
        self.image_dir = data_path / "images"
        self.mask_dir = data_path / "masks"
        self.metadata = pd.read_csv(data_path / metadata_file)

    def __getitem__(self, index):
        image = Image.open(self.image_paths[index]).convert("RGB")
        mask = Image.open(self.mask_paths[index]).convert("L")
        return image, mask
```

11 Common Debugging Issues

Query

Why does my mask overlay not appear as expected? The overlay looks like the original image without any changes.

Answer

This often occurs because the mask is not binary. To ensure proper overlay, you need to explicitly binarize the mask:

```python
mask_np = mask.squeeze().numpy()
binary_mask = (mask_np > 0.5).astype(np.uint8)
```

Listing 3: Binarizing the Mask

Query

Why do I get an `ImportError` when trying to import my module in a notebook?

Answer

Make sure that the `src/` directory is added to `sys.path` at the beginning of your notebook:

```python
import sys
from pathlib import Path

sys.path.append(str(Path("../src").resolve()))
```

Listing 4: Adding src to sys.path

This allows proper module imports like:

```python
from project_name.data import visualize_raw_data
```

12 Git/GitHub and CI/CD Setup

Query

How do I pull everything from the main branch into my branch explainer-Notebook?

Answer

Ensure you're on the correct branch, then pull the latest changes from main:

```
git checkout explainerNotebook
git pull origin main
```

To rebase instead of merging (for a cleaner history):

```
git checkout explainerNotebook
git fetch origin
git rebase origin/main
```

Query

How do I push changes to the explainerNotebook branch?

Answer

Follow these steps to push your changes:

```
git checkout explainerNotebook
git add .
git commit -m "Add/update explainer notebook content"
git push origin explainerNotebook
```

13 Citations and References

Query

How do I cite Weights & Biases (W&B)?

Answer

APA style:

Weights & Biases. (n.d.). *Weights & Biases Documentation.* Retrieved April 23, 2025, from https://wandb.ai/site/

FROM DOWNWARDS THE PROMPTS ORIGINATE FROM ANOTHER CHAT HISTORY.

14 Baseline Segmentation Model

Query

Help me make a baseline for my project that is a segmentation model to detect forest cover in satellite images. The data consists of RGB matrices and a ground truth mask. I want the baseline to check each pixel value and return a 1 in the predicted mask if the pixel value is more green than red and blue. Also include a threshold as input that can be adjusted to say how much larger the green value needs to be in order for it to be predicted as a tree.

Answer

Below is a simple baseline segmentation model implementation based on pixel-wise green dominance:

```
def green_tree_detector(image_batch, threshold=0.1):
    """
    Simple baseline that detects trees based on green channel values.
    Args:
        image_batch: Tensor of shape [batch_size, 3, 256, 256]
        threshold: How much greener a pixel must be compared to other channels
    Returns:
        Tensor of shape [batch_size, 1, 256, 256] with binary tree mask
    """
    batch_size = image_batch.shape[0]
```

```python
    device = image_batch.device

    r = image_batch[:, 0]
    g = image_batch[:, 1]
    b = image_batch[:, 2]

    tree_mask = ((g > (r + threshold)) & (g > (b + threshold))).float()
    return tree_mask.unsqueeze(1)
```

15 Evaluation of the Baseline

Query

Evaluate the baseline model using common metrics.

Answer

The following function evaluates the model using accuracy, precision, recall, F1 score, and IoU:

```python
def evaluate_detector(dataloader, threshold=0.1):
    device = next(iter(dataloader))[0].device

    metrics = {'accuracy': 0, 'precision': 0, 'recall': 0, 'f1': 0, 'iou': 0}
    total_pixels = 0

    for images, masks in tqdm(dataloader, desc="Evaluating baseline detector"):
        predictions = green_tree_detector(images, threshold=threshold)
        pred_binary = (predictions > 0.5).float()

        pred_flat = pred_binary.cpu().numpy().flatten().astype(int)
        mask_flat = masks.cpu().numpy().flatten().astype(int)

        batch_pixels = pred_flat.shape[0]
        total_pixels += batch_pixels

        metrics['accuracy'] += accuracy_score(mask_flat, pred_flat) * batch_pixels
        metrics['precision'] += precision_score(mask_flat, pred_flat, zero_division=0) * batch_pixels
        metrics['recall'] += recall_score(mask_flat, pred_flat, zero_division=0) * batch_pixels
```

```
        metrics['f1'] += f1_score(mask_flat, pred_flat, zero_division=0) * batch_pixels
        metrics['iou'] += jaccard_score(mask_flat, pred_flat, zero_division=0) * batch_pixels

    for key in metrics:
        metrics[key] /= total_pixels

    return metrics
```

16 Visualization Function

Query

Now help me make a visualization function that can visualize the original image, the predicted mask and the original ground truth mask.

Answer

The following function visualizes and saves predictions:

```
def visualize_baseline_predictions(val_loader, device, green_tree_detector, threshold=0.01, num_samples=5):
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    os.makedirs("reports/figures", exist_ok=True)
    save_path = f"reports/figures/baseline_mask_comparison_{timestamp}.png"

    images, true_masks = next(iter(val_loader))
    images, true_masks = images.to(device), true_masks.to(device)

    with torch.no_grad():
        predicted_masks = green_tree_detector(images, threshold=threshold)
        predicted_masks = (predicted_masks > 0.5).float()

    images = images.cpu().numpy().transpose(0, 2, 3, 1)
    true_masks = true_masks.cpu().numpy().squeeze(1)
    predicted_masks = predicted_masks.cpu().numpy().squeeze(1)

    fig, axes = plt.subplots(num_samples, 3, figsize=(10, num_samples * 3))
    for i in range(num_samples):
```

```
        axes[i, 0].imshow((images[i] * 0.5) + 0.5)
        axes[i, 0].set_title("Original Image")
        axes[i, 0].axis("off")

        axes[i, 1].imshow(true_masks[i], cmap="gray")
        axes[i, 1].set_title("Ground Truth Mask")
        axes[i, 1].axis("off")

        axes[i, 2].imshow(predicted_masks[i], cmap="gray")
        axes[i, 2].set_title(f"Baseline Mask (threshold={threshold:.2f})")
        axes[i, 2].axis("off")

    plt.tight_layout()
    plt.savefig(save_path)
    print(f"[INFO] Baseline mask comparison saved as {save_path}")
    plt.show()
```

17 Finding High-Forest Samples

Query

Now help me create a function that takes in our dataloader function and finds the images in the data with the most forest cover in the ground truth mask.

Answer

The following function finds the top-k samples with the most forest pixels:

```
def get_top_forest_ids(
    dataset_name: str,
    data_path: Path,
    metadata_file: str = None,
    image_dir: Path = None,
    label_dir: Path = None,
    img_dim: int = 256,
    top_k: int = 5
):
    transform = transforms.Compose([
        transforms.Resize((img_dim, img_dim)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5]*3, std=[0.5]*3)
    ])
    target_transform = transforms.Compose([
        transforms.Resize((img_dim, img_dim)),
```

```python
        transforms.ToTensor()
    ])

    if dataset_name == "forest":
        dataset = ForestDataset(
            data_path=data_path,
            metadata_file=metadata_file,
            transform=transform,
            target_transform=target_transform
        )
    elif dataset_name == "roboflow":
        dataset = RoboflowTreeDataset(
            image_dir=image_dir,
            label_dir=label_dir,
            patch_size=img_dim,
            transform=transform,
            target_transform=target_transform
        )
    else:
        raise ValueError("dataset_name must be either 'forest' or 'roboflow'")

    forest_coverage = []
    for idx in range(len(dataset)):
        _, mask = dataset[idx]
        forest_pixels = (mask > 0.5).sum().item()
        forest_coverage.append((idx, forest_pixels))

    top_ids = sorted(forest_coverage, key=lambda x: x[1], reverse=True)[:top_k]
    return [idx for idx, _ in top_ids]
```

18 Basic Augmentation Strategies

Query

I want to augment our data when training, what are general ways to augment an image?

Answer

Here are standard augmentation techniques useful for training:

- **Horizontal flip**:

```python
torch.flip(image, [2])
```

- **Vertical flip**:

```
torch.flip(image, [1])
```

- **Random rotation** (by 90°, 180°, 270°):

```
k = random.choice([1,2,3])
torch.rot90(image, k, [1, 2])
```

19 API CHAT history

https://chatgpt.com/share/680b6dd4-7094-8004-8763-e74c8cf81a92